High Throughput Computing at NERSC and Beyond

Photo: Michael Wolf, Mother Jones Magazine

Stephen Bailey
Amy Nesky
Physics
Dan Gunter
Monte Goode
CRD
Damir Sudar
Life Sciences
Anubhav Jain
Kristen Persson
EETD
David Skinner
Shane Cannon
NERSC

3D Map of the Universe

Stephen Bailey
LBNL

Current Generation

– 2.5M galaxies, 2600 jobs, 50k CPU-hours

Next Generation

– 25M galaxies, 100k (?) jobs, 10M CPU-hours

How to adapt this workflow to NERSC-level supercomputers?

Common Problem: #tasks >> #jobs

Solutions tend to be...

- One-off solutions for single projects, or
- Overly complex, or
- Too simple

Those aren't bad solutions

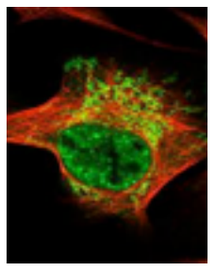
- e.g. PanDA, taskfarmer

They just don't quite fill the right niche

- David Skinner: "Tactical High Throughput Computing"
- Lightweight & flexible to deploy, setup, teardown
- In the spirit of Unix command line tools
  - toolkit to build what you need
  - not monolithic Swiss army knife workflow app
- Human performance/scaling is part of the optimization

High Throughput Computing LDRD

BOSS / DESI
Flagship DOE cosmology survey studying Dark Energy:
Millions of spectra



Protein Atlas
Mapping and understanding proteins within the cell:
Millions of images

Materials Project
Simulating next-gen materials to address energy needs:
Many thousands of simulations

3 Examples / 3 Divisions
there are many more

Need NERSC-scale resources to process 1k–1M tasks

User Requirements

Scaling

- Load ~1000 tasks/second, up to ~1M tasks per workflow
- Minimal overhead when processing >10 minute tasks
- ~1000 simultaneous workflows
- *~1000 simultaneous workers each [in progress]*

Features

- Dependencies & priorities between tasks
- Python + command line (Goal: REST API)
- *Parallelism within tasks [in progress]*

Usability

- Easy to deploy and use
- Integrates with *existing* clusters & super-computers

FireWorks

http://pythonhosted.org/FireWorks/

Developed by Anubhav Jain, Materials Project

– Mature, full featured, battle tested

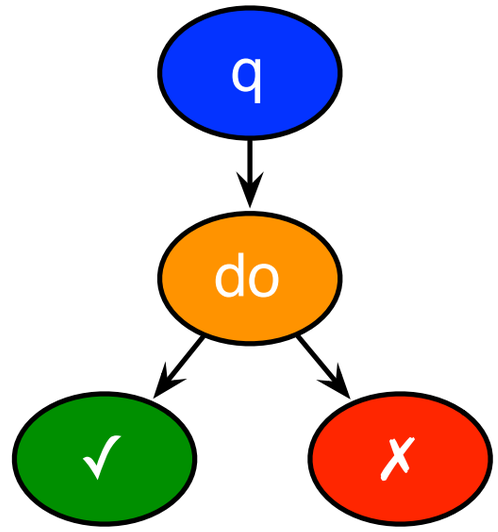
Try same workflows on both

– possible? easy? fast?

– already good enough?

Fix bottlenecks for both

Add features to qdo



https://bitbucket.org/berkeleylab/qdo

Developed by SB, BOSS/DESI

– The scrappy young upstart

– Focus on simplicity, flexibility

Goal for qdo

If it is possible with qdo,
then it is easier with qdo than anything else

StephenBailey@lbl.gov

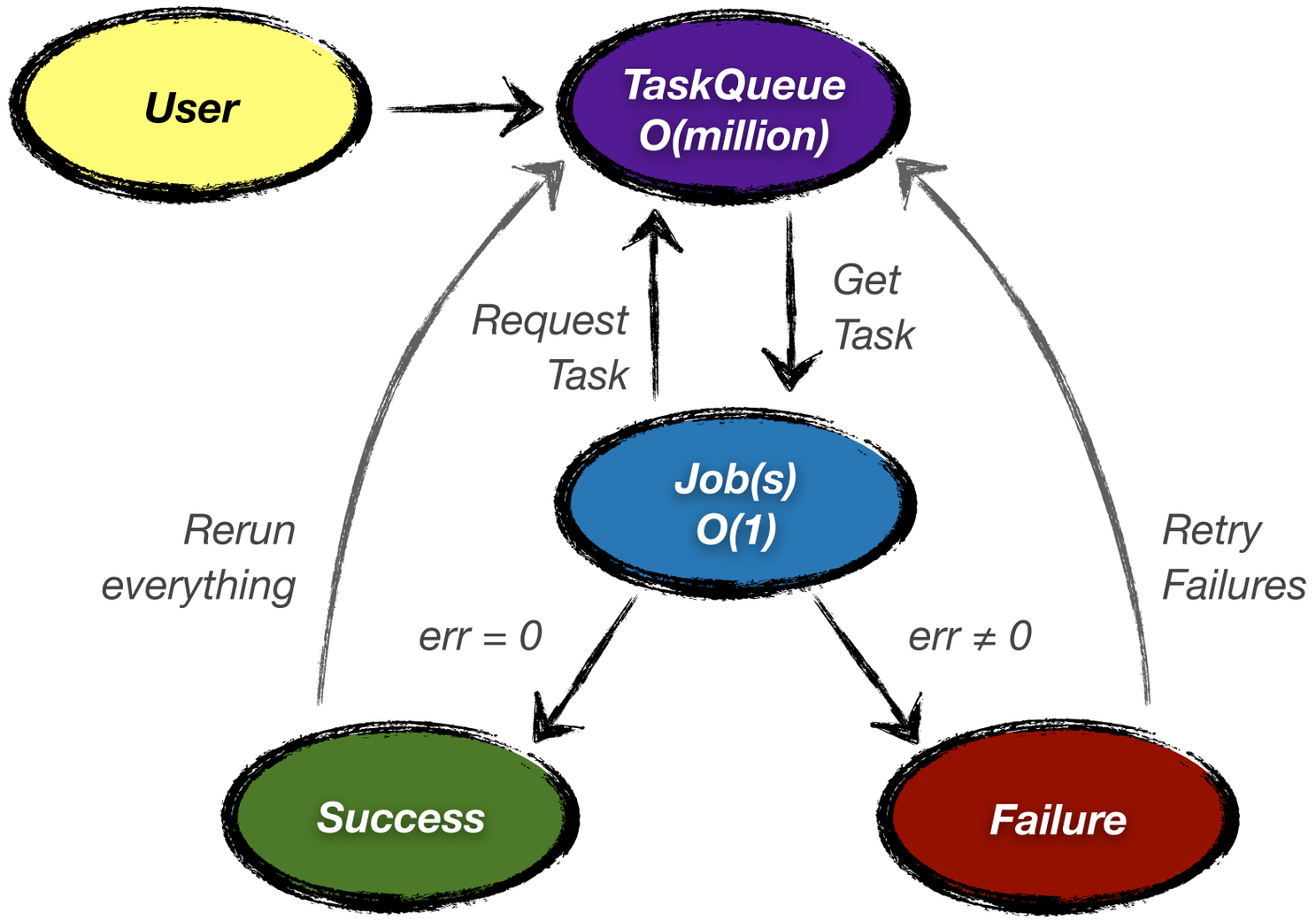
Not easier?

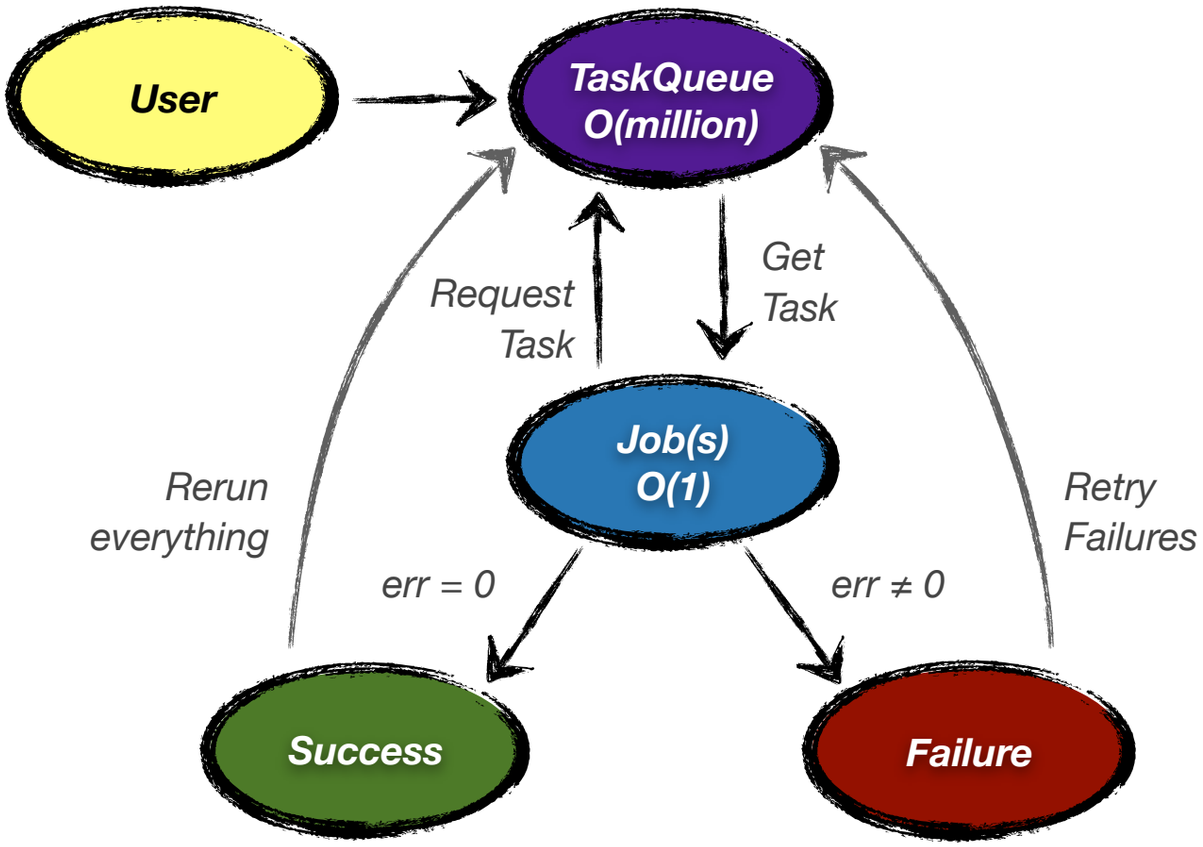
- Let me know why
- This is a high priority for qdo

Not possible?

- Let me know why
- But no promises
  - I would rather have 80% of cases be easy than 100% of cases be possible

qdo Model

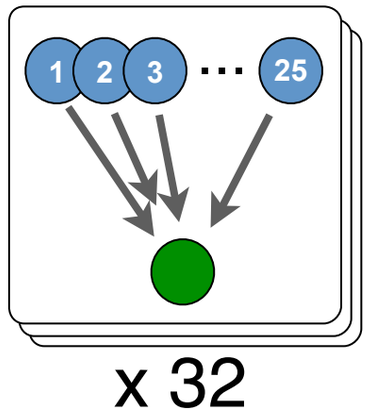

User
TaskQueue
O(million)
Request
Task
Get
Task
Job(s)
O(1)
Rerun
everything
Retry
Failures
err = 0
err ≠ 0
Success
Failure



Key Features

- # tasks >> # batch jobs
- Flexibility
  - Scale up/down # workers
  - Add tasks after jobs have started
- Robustness[*]
  - Queue independent of job & task failures
  - Retry just the failures
- Manage tasks in aggregate
  - Progress stats while workers are running
  - Only deal with an individual task if something went wrong with it

Real world example



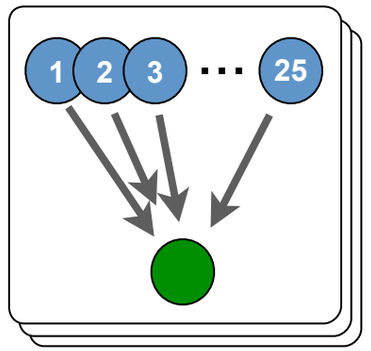

```
#- Create a queue
import qdo
q = qdo.create(“extract”)

#- Make groups of commands to run
for group in range(32):
    commands = list()
    for i in range(25):
        commands.append(“extract {} {}”.format(group, i)

    #- Add commands to queue
    ids = q.add_multiple(commands)
    q.add(“merge ”+str(g), requires=ids)

#- Launch 50 jobs to process the 832 tasks
q.launch(50)
```

Real world example



x 32

```
#- Check status: command line interface
#- (same info available via python interface)
[sbailey]$ qdo list
QueueName         State   Waiting  Pending  Running Succeeded   Failed
extract           Active        0        0        0       826        6

#- 6 Failed!?!  Which ones?
[sbailey]$ qdo tasks extract --state Failed
State       Task
FAILED      extract 3 18
FAILED      merge 3
...

#- Debug, fix some stuff, then rerun just the failed tasks
[sbailey]$ qdo retry extract
6 tasks reset to pending

[sbailey]$ qdo launch extract 3
...
```

Priorities

```
#- Load 1000 tasks
import qdo
q = qdo.create('Analyze')
for i in range(1000):
    q.add(‘analyze -n ’+str(i))

q.launch(10)
```

```
#- after awhile (even from another process)
q.add(‘calibrate blat.dat’, priority=100)
```

Advanced: params instead of execs

```
#- Can add any JSON-able object
q.add( dict(a=1, b=2) )
q.add( dict(a=3, b=4) )
q.add( dict(a=5, b=6) )

#- Pass in a template script to be expanded with options
q.launch(1, script="analyze -a {a} -b {b}")
```

```
#- Or pass in a function that takes task as input
def func(params):
    result = params['a'] + params['b']
    print "a+b = {}".format(result)

q.do(func=func)
```

Roadmap

Deploy to other users

- Get others to try it. Does it “stick”?

Web interface

- REST API + Webpage GUI

Robustness

- orphaned “running” jobs if batch job hits wallclock limit
- optional auto-retry of “random” failures

Performance tuning

Features

- dependencies: required to *finish* vs. required to *succeed*
- parallel tasks
- easier stdout/stderr log management

Summary

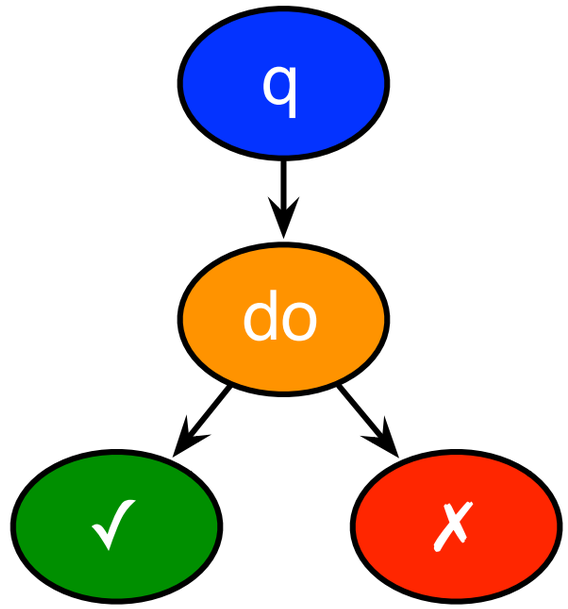
StephenBailey@lbl.gov

Tactical high throughput computing

- So easy you don't need an expert to help you write your workflow
- qdo enables O(1M) tasks within standard batch job framework

Metric for workflow tools

- What fraction of user's mental energy is spent on workflow vs. underlying algorithms?



Getting qdo

- https://bitbucket.org/berkeleylab/qdo
- At NERSC

```
module use /project/projectdirs/cosmo/software/modules/carver/
module load qdo/0.5
qdo --help
pydoc qdo
```

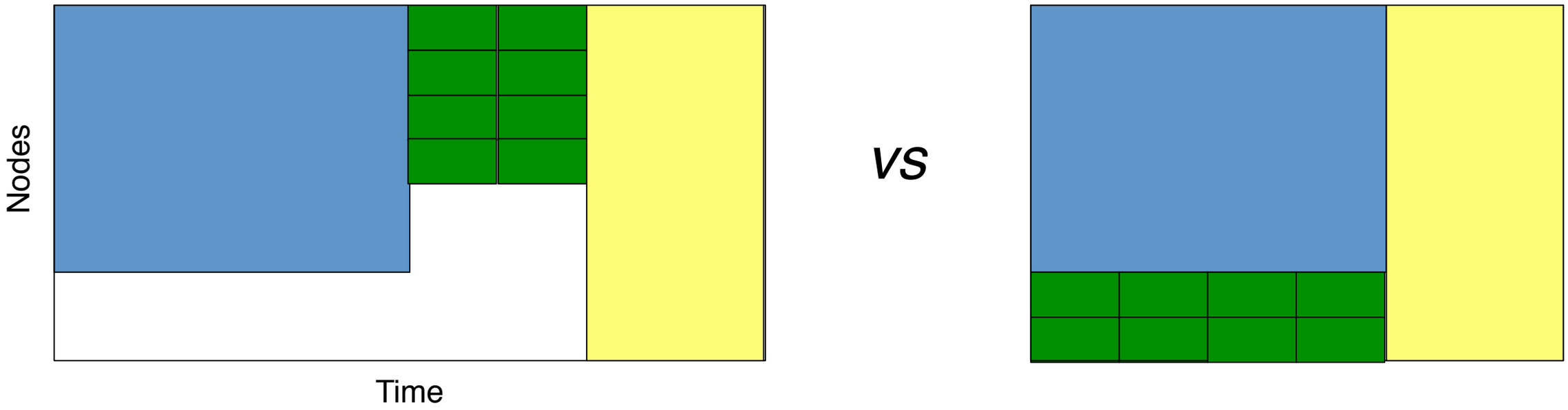
[Hopper & Edison coming soon]

Backup Slides

Efficient Queuing

HTC jobs can be flexible in shape

- Currently user has to pick both width and length
- Could just specify area and constraints instead
- Let system pick the most efficient packing



Auto-optimize:

— Easier for user

— More efficient for overall queue

— Easier said than done, i.e. R&D problem

Example: run a single command

```
#- Command line
qdo add Blat “analyze blat.dat”   #- creates queue & adds cmd
qdo launch Blat 1                 #- launches 1 batch job

#- Python
import qdo
q = qdo.create(“Blat”)            #- creates queue
q.add(“analyze blat.dat”)         #- adds command
q.launch()                        #- launches 1 batch job
```

Example: run multiple commands

```
#- Command line
qdo load Blat commands.txt      #- loads file with commands
qdo launch Blat 24 --pack       #- 1 batch job; 24 mpi workers

#- Python
import qdo
q = qdo.create("Blat")
for i in range(1000):
    q.add("analyze blat{}.dat".format(i))
q.launch(24, pack=True)

#- Python load 1M tasks
commands = list()
for x in range(1000):
    for y in range(1000):
        commands.append("analyze -x {} -y {}".format(x, y))

q.add_multiple(commands)   #- takes ~2 minutes
q.launch(1024, pack=True)
```

Example: what queues exist?

```
#- Command line
qdo list
QueueName     State   Waiting  Pending  Running Succeeded    Failed
BlatFoo       Active        0        3        0         0         0
EchoChamber   Active        0       46        0        50         4

#- Python
qdo.qlist()
[<qdo.Queue BlatFoo at 0x1087cc790>,
 <qdo.Queue EchoChamber at 0x1087cc810>]

print qdo.qlist()[0]
BlatFoo is Active
Waiting    : 0
Pending    : 3
Running    : 0
Succeeded  : 0
Failed     : 0
```

Example: check status

```
#- Command line
qdo status EchoChamber
EchoChamber is Active
Waiting    : 0
Pending    : 46
Running    : 0
Succeeded  : 50
Failed     : 4

qdo tasks EchoChamber
qdo tasks EchoChamber --state=Failed
State       Task
Failed      echo 13 && sleep 1
Failed      echo 15 && sleep 1
Failed      echo 19 && sleep 1
Failed      echo 29 && sleep 1
(they failed b/c I killed them
while running)
```

```
#- Python
q = qdo.connect("EchoChamber")
q.status()
{'name': 'EchoChamber',
 'ntasks': {'Failed': 4,
  'Pending': 46,
  'Running': 0,
  'Succeeded': 50,
  'Waiting': 0},
  'state': u'Active',
  'user': 'sbailey'}

#- list of dicts of tasks
q.tasks(state='Failed')
```

Example: dependencies

```
import qdo
q = qdo.create('MapReduce')
commands = ['echo hello '+str(i) for i in range(10)]

#- Adding commands returns list of task IDs
taskids = q.add_multiple(commands)

#- Use that list to set dependencies
q.add('echo goodbye', requires=taskids)
```

Example: retry, rerun, recover

```
import qdo
q = qdo.connect('Blat')

#- Retry just the failed tasks
q.retry()

#- Rerun everything
q.rerun()

#- Recover from failed jobs leaving "running" tasks behind
q.recover()

#- Same thing from command line
qdo retry Blat
qdo rerun Blat --force    #- requires "--force" for safety
qdo recover Blat
```